

Práctica 1 Aprendizaje Automático

Antonio Álvarez Caballero

Generación y visualización de datos

Ejercicio 1

En primer lugar, creamos una función que crea un `data.frame` con valores aleatorios según una distribución uniforme. N es el número de filas del `data.frame`, dim el número de columnas y $rango$ el rango donde estarán los valores.

```
simula_unif <- function(N, dim, rango){  
  res <- data.frame(matrix(nrow = N, ncol = dim))  
  
  for(i in 1:N){  
    res[i,] <- runif(dim, rango[1], rango[length(rango)])  
  }  
  
  names(res) <- c("X", "Y")  
  
  return(res)  
}
```

Ejercicio 2

Del mismo modo creamos una función análoga para la distribución *normal* o *gaussiana*.

```
simula_gauss <- function(N, dim, sigma){  
  res <- data.frame(matrix(nrow = N, ncol = dim))  
  
  for(i in 1:N){  
    res[i,] <- rnorm(dim, sd = sqrt(sigma))  
  }  
  
  names(res) <- c("X", "Y")  
  
  return(res)  
}
```

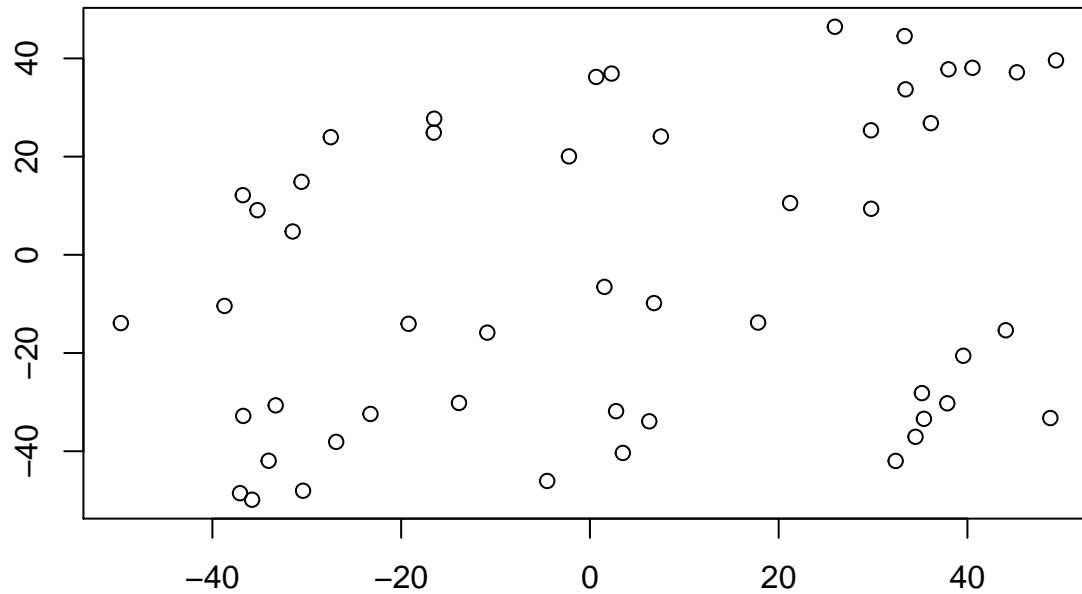
Ejercicios 3 y 4

Ahora asignamos los resultados a sendos objetos del tipo *data.frame* y las dibujamos.

```
muestra.uniforme <- simula_unif(50, 2, -50:50)  
muestra.gaussiana <- simula_gauss(50, 2, 5:7)
```

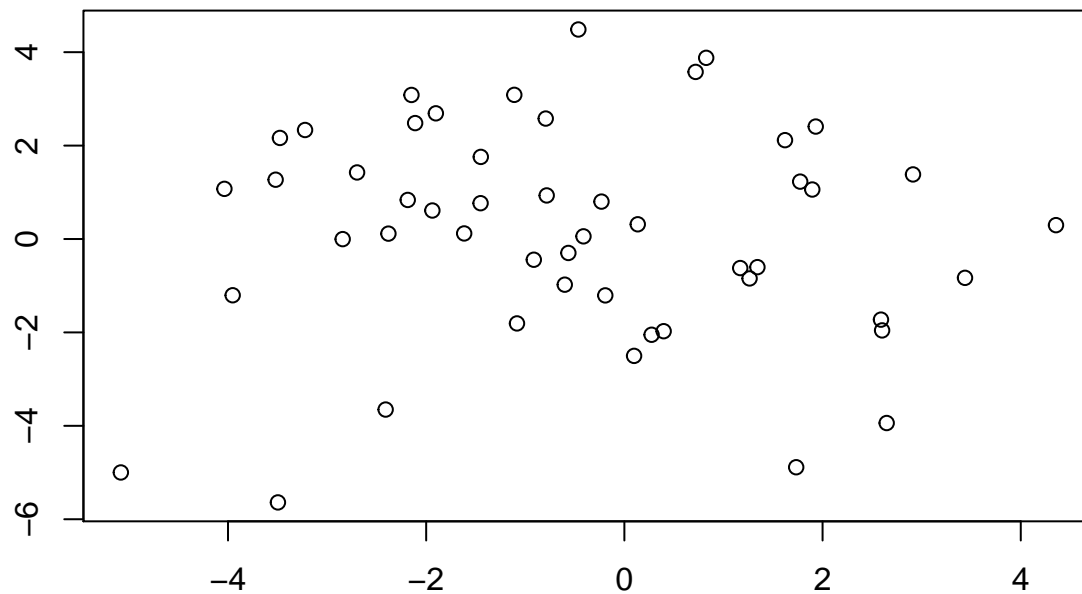
```
plot(muestra.uniforme, main = "Distribución uniforme", xlab = "", ylab = "")
```

Distribución uniforme



```
plot(muestra.gaussiana, main = "Distribución normal", xlab = "", ylab = "")
```

Distribución normal



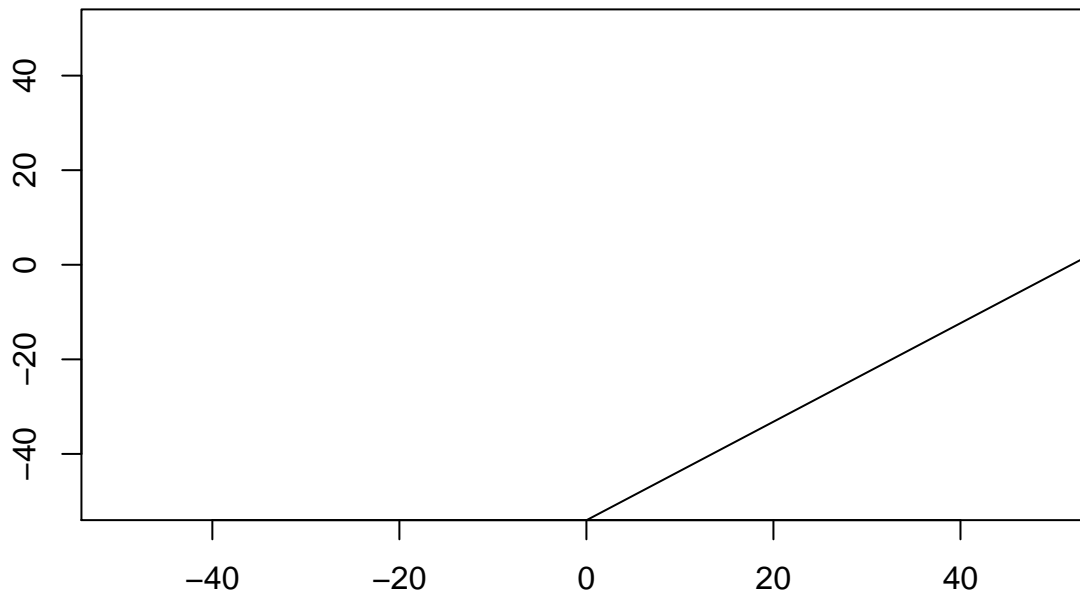
Ejercicio 5

Ahora escribimos una pequeña función para calcular una recta dados dos puntos. Daremos la recta con la ecuación punto pendiente, por lo que tenemos que calcular la pendiente y la desviación.

```
simulaRecta <- function(intervalo){  
  A <- runif(2, intervalo[1], intervalo[length(intervalo)])  
  B <- runif(2, intervalo[1], intervalo[length(intervalo)])  
  
  m <- (A[2] - B[2]) / (A[1] - B[1])  
  b <- A[2] - m * A[1]  
  
  return(c(b,m))  
}
```

Generamos una recta aleatoria en $[-50, 50]$

```
rectaPrueba <- simulaRecta(-50:50)  
  
plot(1, type="n", xlab="", ylab="", xlim=c(-50, 50), ylim=c(-50, 50))  
abline(coef = rectaPrueba)
```



Ejercicio 6

Ahora clasificamos los datos de la muestra uniforme según otra recta aleatoria. Guardamos las muestras clasificadas para próximos ejercicios.

```
rectaClasificacion <- simulaRecta(-50:50)

muestra.uniforme.etiquetada <- cbind(muestra.uniforme, Etiqueta = sign(muestra.uniforme$Y -
  rectaClasificacion[2] * muestra.uniforme$X - rectaClasificacion[1]))

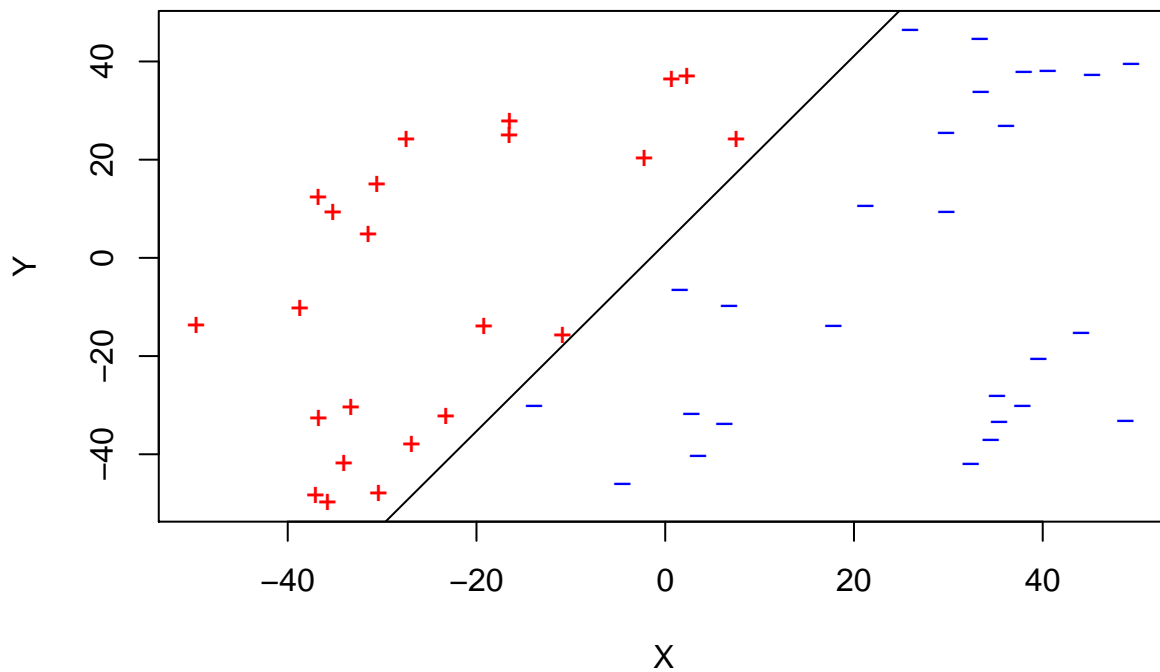
muestra.uniforme.positiva <- subset(muestra.uniforme.etiquetada, Etiqueta ==
  1)

muestra.uniforme.negativa <- subset(muestra.uniforme.etiquetada, Etiqueta ==
  -1)

formas <- ifelse(muestra.uniforme.etiquetada$Etiqueta == 1, "+", "-")
colores <- ifelse(muestra.uniforme.etiquetada$Etiqueta == 1, "red", "blue")

plot(muestra.uniforme, pch = formas, col = colores)

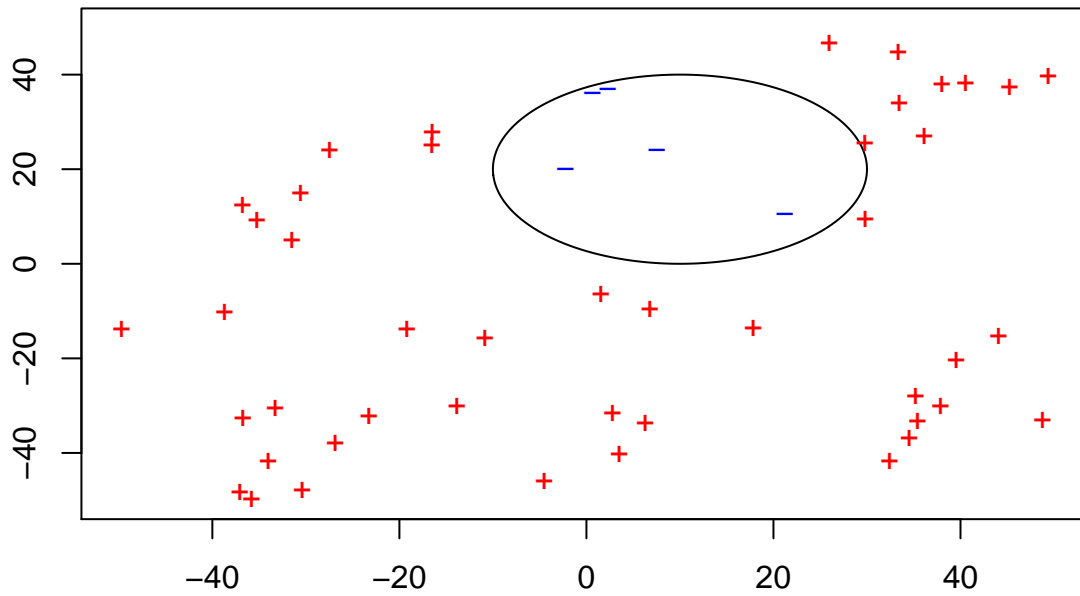
abline(coef = rectaClasificacion)
```



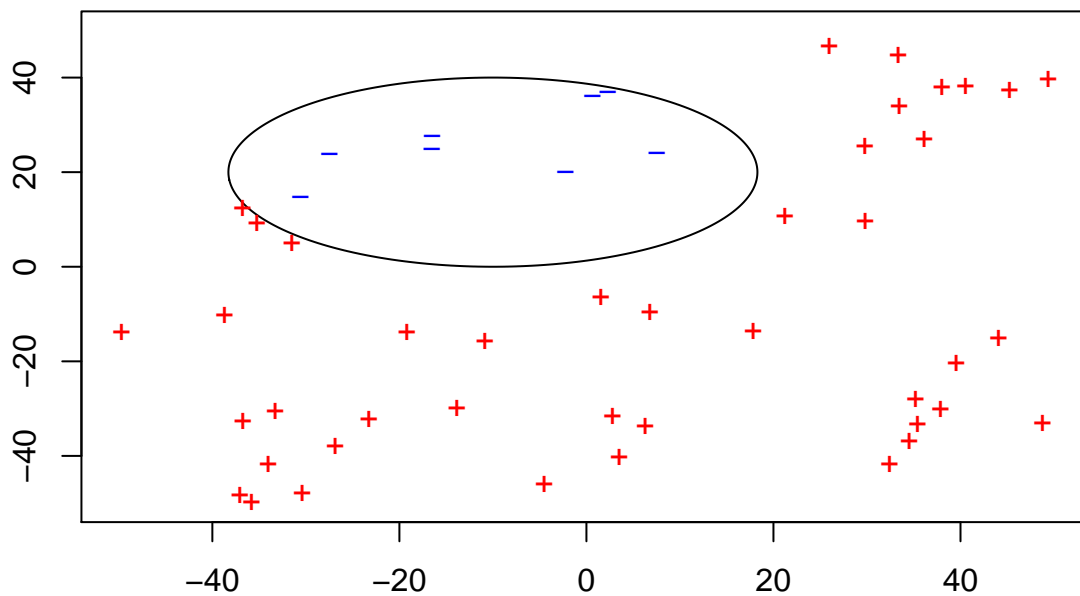
Ejercicio 7

Una vez hemos clasificado los datos con una recta, clasificamos con otro tipo de funciones.

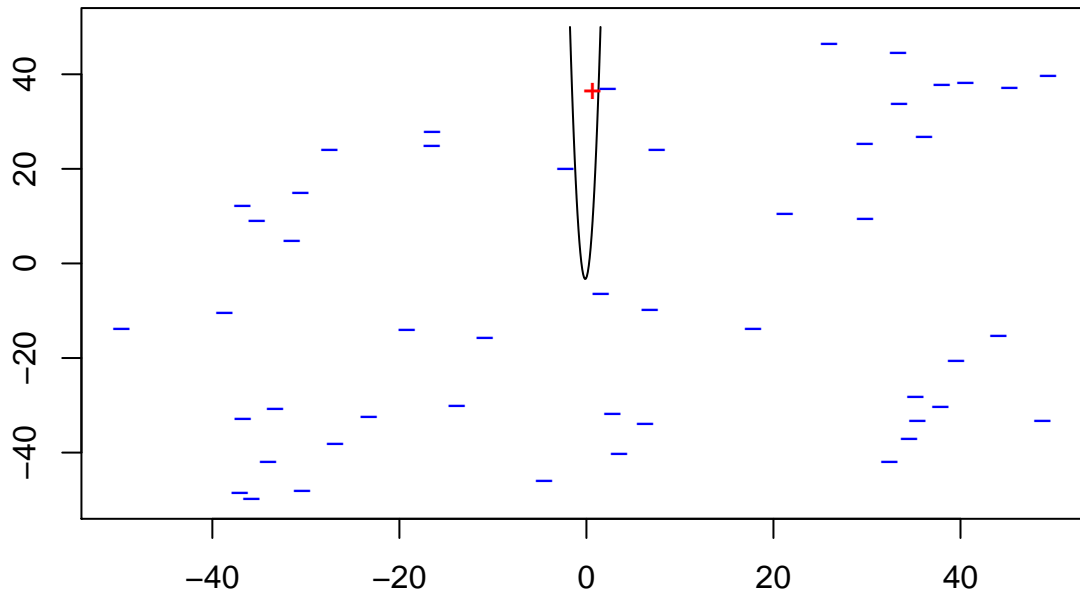
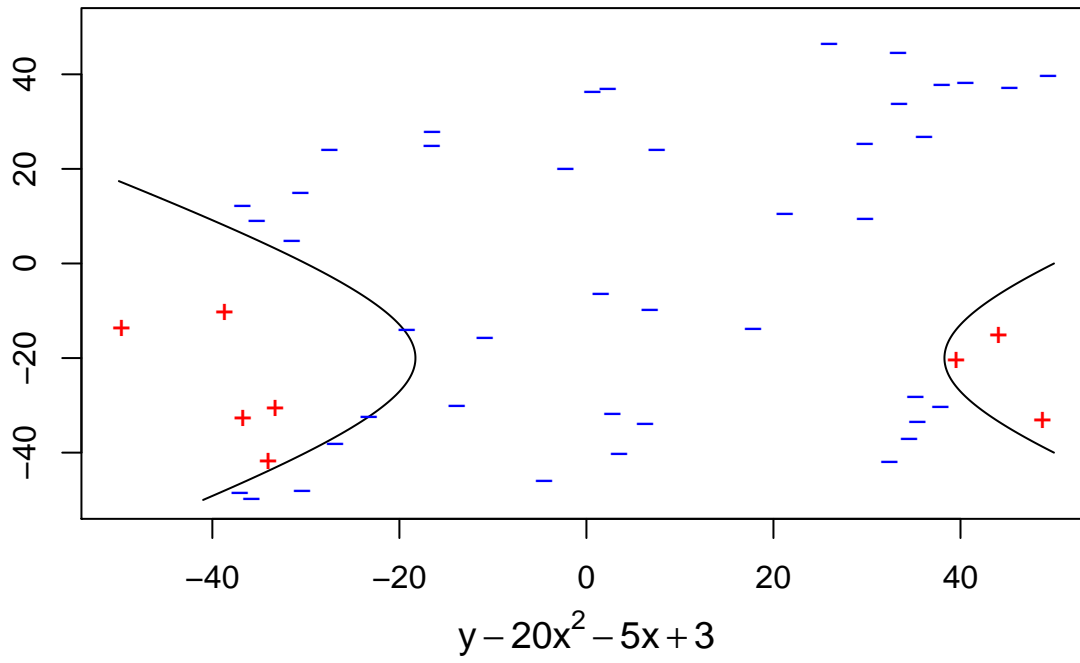
$$(x-10)^2 + (y-20)^2 - 400$$



$$0.5(x+10)^2 + (y-20)^2 - 400$$



$$0.5(x - 10)^2 - (y + 20)^2 - 400$$



Añadir aquí conclusiones de las regiones nuevas comparadas con la lineal.

Ejercicio 8

Tomamos la muestra clasificada y creamos un vector de booleanos con el 10% *TRUE* y el restante *FALSE*. Así luego extraeremos una porción de la muestra.

```
# Tomamos la porción deseada
porcion <- 10
numero.datos.positivos <- ceiling(nrow(muestra.uniforme.positiva) * porcion/100)
numero.datos.negativos <- ceiling(nrow(muestra.uniforme.negativa) * porcion/100)

vector.aleatorio.muestras.positivas <- c(rep(FALSE, numero.datos.positivos),
    rep(TRUE, nrow(muestra.uniforme.positiva) - numero.datos.positivos))

vector.aleatorio.muestras.negativas <- c(rep(FALSE, numero.datos.negativos),
    rep(TRUE, nrow(muestra.uniforme.negativa) - numero.datos.negativos))

vector.aleatorio.muestras.positivas <- sample(vector.aleatorio.muestras.positivas)
vector.aleatorio.muestras.negativas <- sample(vector.aleatorio.muestras.negativas)

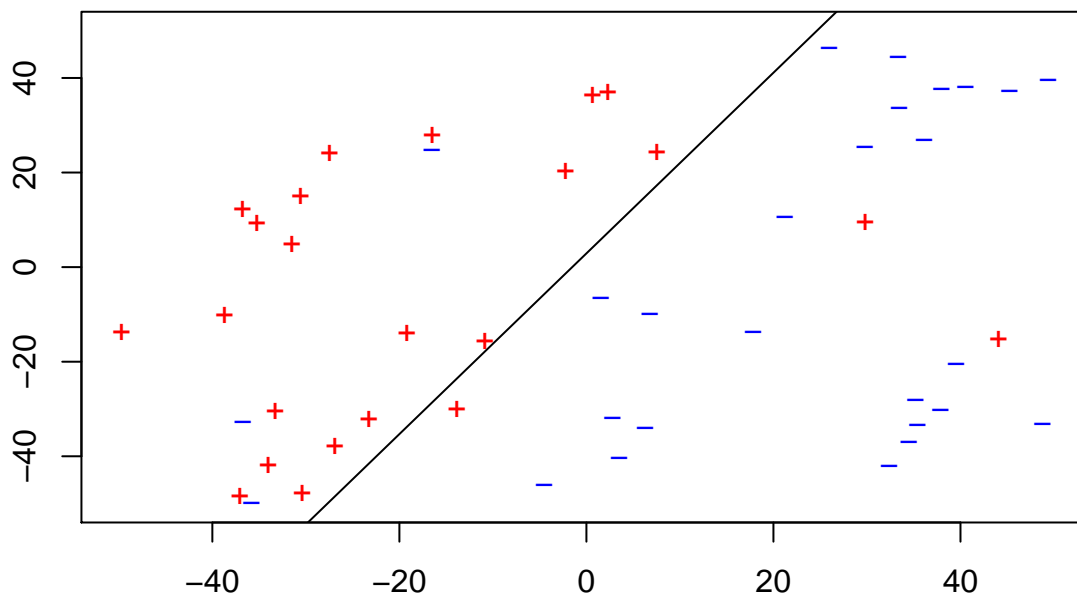
muestra.uniforme.positiva.nueva <- muestra.uniforme.positiva[vector.aleatorio.muestras.positivas,
    ]
muestra.uniforme.negativa.nueva <- muestra.uniforme.negativa[vector.aleatorio.muestras.negativas,
    ]

muestra.uniforme.positiva.nueva <- rbind(muestra.uniforme.positiva.nueva,
    muestra.uniforme.negativa[!vector.aleatorio.muestras.negativas, ])

muestra.uniforme.negativa.nueva <- rbind(muestra.uniforme.negativa.nueva,
    muestra.uniforme.positiva[!vector.aleatorio.muestras.positivas, ])

muestra.uniforme.positiva.nueva$Etiqueta <- 1
muestra.uniforme.negativa.nueva$Etiqueta <- -1
```

Ahora mostramos los nuevos resultados



Ajuste del algoritmo Perceptron (PLA)

Ejercicio 1

Vamos a escribir una función que implemente el algoritmo *Perceptron* o *PLA*. *Datos* es un *data.frame* con cada muestra por fila, *label* el vector de etiquetas, el cual debe tener la misma longitud que los datos de muestra, *max_iter* es el número máximo de iteraciones del algoritmo y *vini* es el vector inicial, que debe tener la misma dimensión que los datos.

```
ajusta_PLA <- function(datos, label, max_iter, vini){
  cambio <- TRUE
  w <- vini
  iteraciones <- 0
  errores <- 0
  X <- cbind(1,datos)

  while (cambio && iteraciones < max_iter){
    cambio <- FALSE
    errores <- 0
    for (i in 1:nrow(datos)){
      x_i <- as.numeric(X[i,])
      prodEscalar <- crossprod(w,x_i)
      if (sign(prodEscalar) != label[i]){
        cambio <- TRUE
        errores <- errores + 1
        w <- w + label[i] * x_i
      }
    }
    iteraciones <- iteraciones + 1
  }

  resultado <- list("Peso inicial" = vini, "Pesos" = w,
                  "Iteraciones" = iteraciones, "Errores" = errores,
                  "Recta" = c(-w[1]/w[3], -w[2]/w[3]) )
  return (resultado)
}
```


Ejercicio 2

Vamos a hacer una simulación de prueba. Tomamos los datos de la muestra uniforme y un vector de etiquetas aleatorio.

```
etiquetas <- t(as.vector(muestra.uniforme.etiquetada["Etiqueta"]))
vini <- rep(0,3)

resultado <- ajusta_PLA(muestra.uniforme, etiquetas, 100, vini)

w <- as.vector(resultado$Pesos)
iteraciones <- as.numeric(resultado$Iteraciones)

print(iteraciones)
```

```
## [1] 3
```

```
mediaIteraciones <- 0

for (i in 1:10){
  vini <- runif(3, 0,1)
  print(vini)
  resultado <- ajusta_PLA(muestra.uniforme, etiquetas, 100, vini)
  mediaIteraciones <- mediaIteraciones + as.numeric(resultado$Iteraciones)
  print(resultado$Iteraciones)
}
```

```
## [1] 0.7120449 0.6921819 0.2605972
## [1] 3
## [1] 0.35961356 0.48126674 0.05552475
## [1] 3
## [1] 0.8716329 0.2241864 0.6137978
## [1] 3
## [1] 0.7852743 0.6781014 0.4587150
## [1] 3
## [1] 0.88137107 0.79311452 0.07931007
## [1] 3
## [1] 0.2620294 0.2664431 0.4751918
## [1] 3
## [1] 0.2555615 0.2817887 0.7604727
## [1] 3
## [1] 0.06461388 0.05177023 0.39075008
## [1] 3
## [1] 0.3270345 0.5473843 0.8684330
## [1] 3
## [1] 0.0992748 0.5737763 0.1939429
## [1] 3
```

```
mediaIteraciones <- mediaIteraciones / 10

print(mediaIteraciones)
```

```
## [1] 3
```

De estos resultados podemos decir que si los datos son separables, el *PLA* converge rápidamente.

Ejercicio 3

Vamos a ejecutar el *PLA* con un conjunto de datos no separable.

```
muestra.uniforme.no.separable <- rbind(cbind(muestra.uniforme.positiva.nueva),
                                         cbind(muestra.uniforme.negativa.nueva))

rownames(muestra.uniforme.no.separable) <- NULL

etiquetas <- t(as.vector(muestra.uniforme.no.separable["Etiqueta"]))

vini <- rep(0,3)

resultado <- ajusta_PLA(muestra.uniforme.no.separable[,c("X","Y")], etiquetas, 10, vini)
print(resultado$Errores)
```

```
## [1] 11
```

```
resultado <- ajusta_PLA(muestra.uniforme.no.separable[,c("X","Y")], etiquetas, 100, vini)
print(resultado$Errores)
```

```
## [1] 15
```

```
resultado <- ajusta_PLA(muestra.uniforme.no.separable[,c("X","Y")], etiquetas, 1000, vini)
print(resultado$Errores)
```

```
## [1] 14
```

Visto los resultados, podemos afirmar que sobre un conjunto no separable de datos, el *PLA* no converge y por tanto no da buenas soluciones por muchas iteraciones que le permitamos.

Ejercicio 4

Tomemos la función cuadrática del Ejercicio 7 y realicemos el mismo procedimiento.

```
f <- function(x,y) {(x-10)^2 + (y-20)^2 - 400}

muestra.uniforme.circular <- cbind(muestra.uniforme,
                                   Etiqueta = sign(f(muestra.uniforme$X, muestra.uniforme$Y)))

etiquetas <- t(as.vector(muestra.uniforme.circular["Etiqueta"]))

vini <- rep(0,3)

resultado <- ajusta_PLA(muestra.uniforme.circular[,c("X","Y")], etiquetas, 10, vini)
print(resultado$Errores)
```

```
## [1] 20
```

```
resultado <- ajusta_PLA(muestra.uniforme.circular[,c("X","Y")], etiquetas, 100, vini)
print(resultado$Errores)
```

```
## [1] 10
```

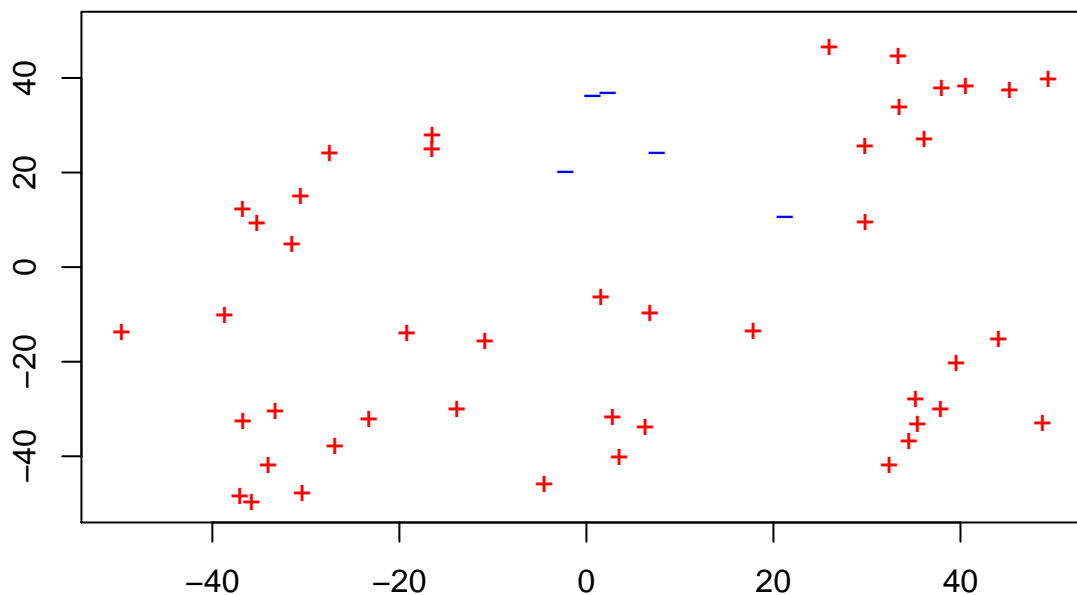
```
resultado <- ajusta_PLA(muestra.uniforme.circular[,c("X","Y")], etiquetas, 1000, vini)
print(resultado$Errores)
```

```
## [1] 8
```

```
plot(1, type="n", xlab="", ylab="", xlim=c(-50, 50), ylim=c(-50, 50))

points(subset(muestra.uniforme.circular,Etiqueta == 1), pch = "+", col = "red")
points(subset(muestra.uniforme.circular,Etiqueta == -1), pch = "-", col = "blue")

abline(coef = resultado$Recta)
```



Con este conjunto de datos, en los que la separabilidad tiene forma circular, el *PLA* no sirve absolutamente de nada: contra más iteraciones, menos clasifica al dejar todos los datos a un lado.

Ejercicio 5

Vamos a modificar la función *ajusta_PLA* para que vaya haciendo plot cada iteración.

```
ajusta_PLA_PLOT <- function(datos, label, max_iter, vini){
  cambio <- TRUE
  w <- vini
  iteraciones <- 0
  errores <- 0
  X <- cbind(1,datos)

  datos.etiquetados <- cbind(datos, Etiqueta = t(label))

  while (cambio && iteraciones < max_iter){
    cambio <- FALSE
    errores <- 0
    for (i in 1:nrow(datos)){
      x_i <- as.numeric(X[i,])
      prodEscalar <- crossprod(w,x_i)
      if (sign(prodEscalar) != label[i]){
        cambio <- TRUE
        errores <- errores + 1
        w <- w + label[i] * x_i
      }
    }
    ### PLOT
    recta <- c(-w[1]/w[3], -w[2]/w[3])
    plot(0, type="n", xlab="", ylab="", xlim=c(-50, 50), ylim=c(-50, 50))

    points(subset(datos.etiquetados,Etiqueta == 1), pch = "+", col = "red")
    points(subset(datos.etiquetados,Etiqueta == -1), pch = "-", col = "blue")

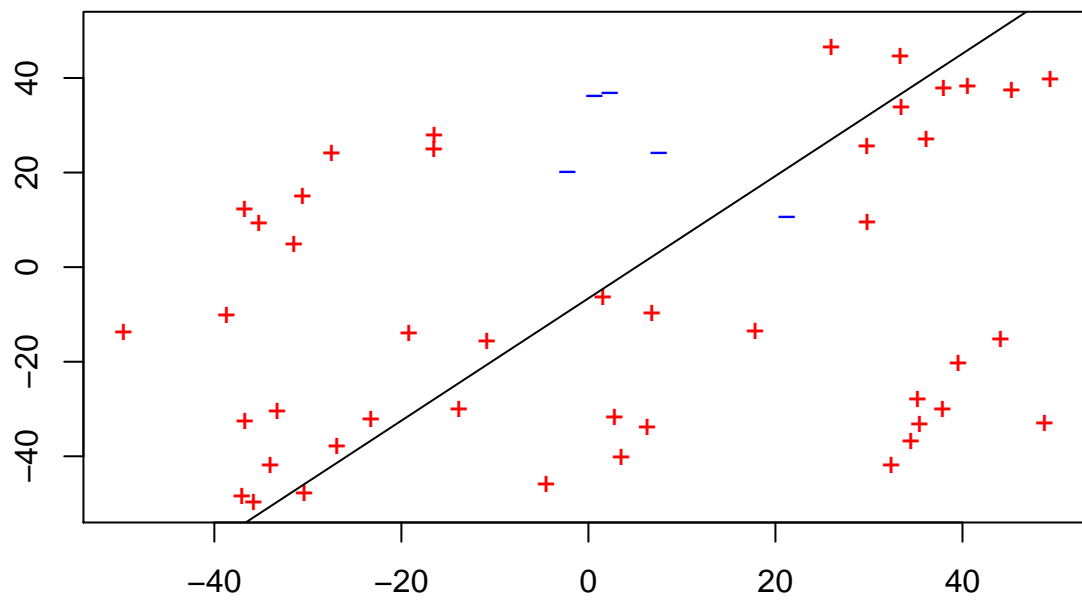
    abline(coef = recta)

    #####
    iteraciones <- iteraciones + 1
    print(paste("Iteracion", iteraciones))
    print(paste("Errores:",errores))
  }

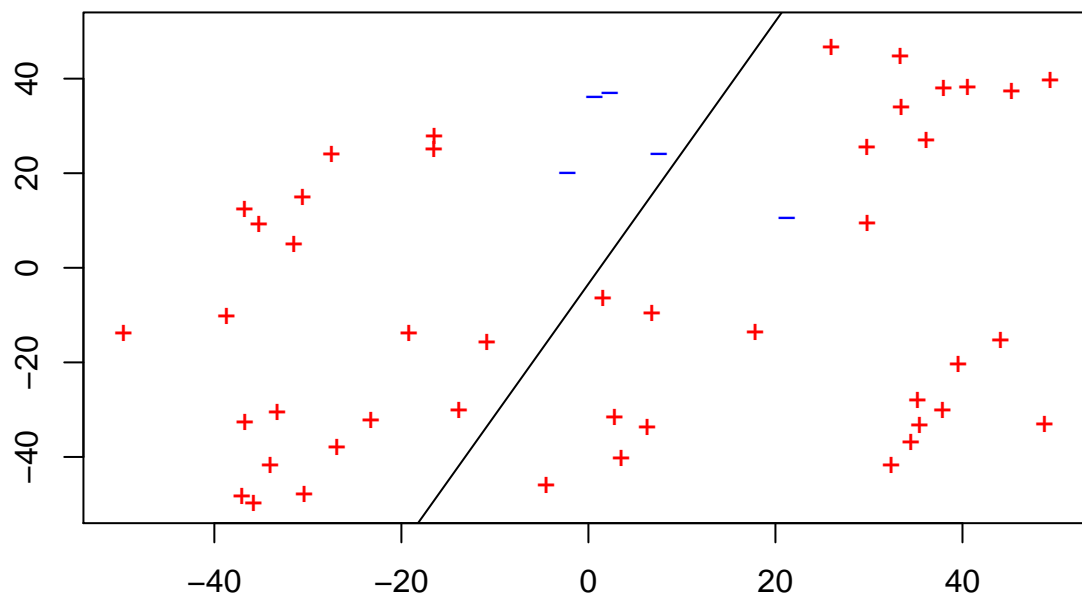
  resultado <- list("Peso inicial" = vini, "Pesos" = w,
                  "Iteraciones" = iteraciones, "Errores" = errores,
                  "Recta" = c(-w[1]/w[3], -w[2]/w[3]) )
  return (resultado)
}

vini <- rep(0,3)

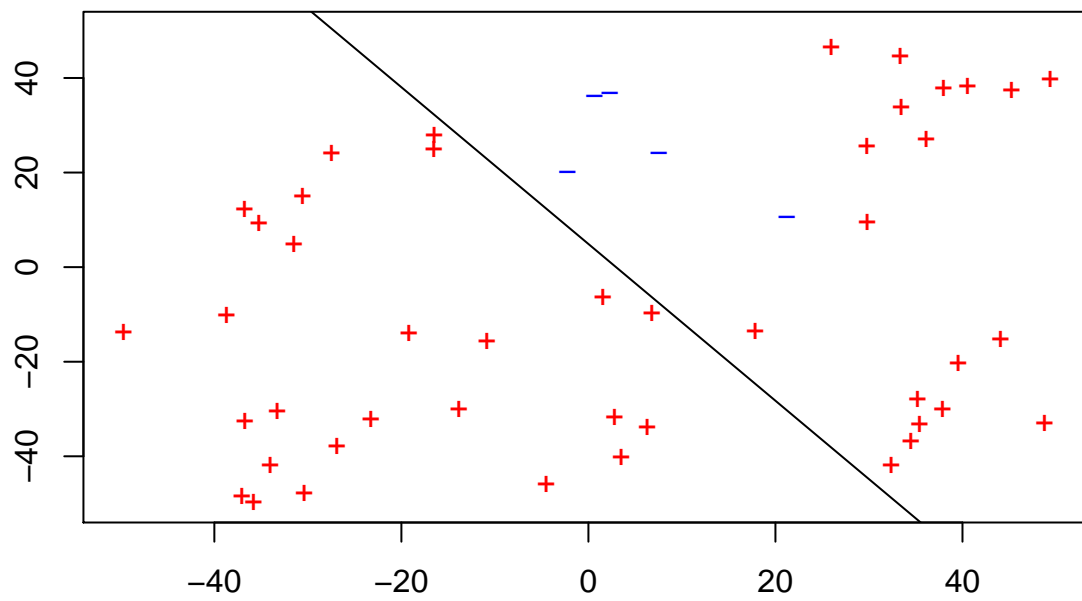
resultado <- ajusta_PLA_PLOT(muestra.uniforme.circular[,c("X","Y")], etiquetas, 15, vini)
```



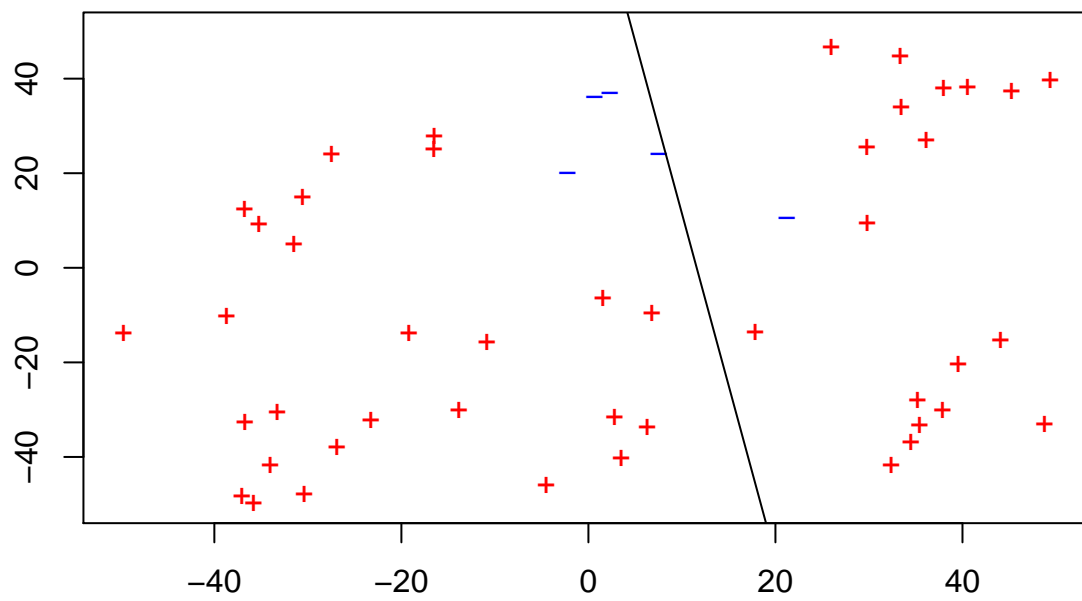
```
## [1] "Iteracion 1"
## [1] "Errores: 26"
```



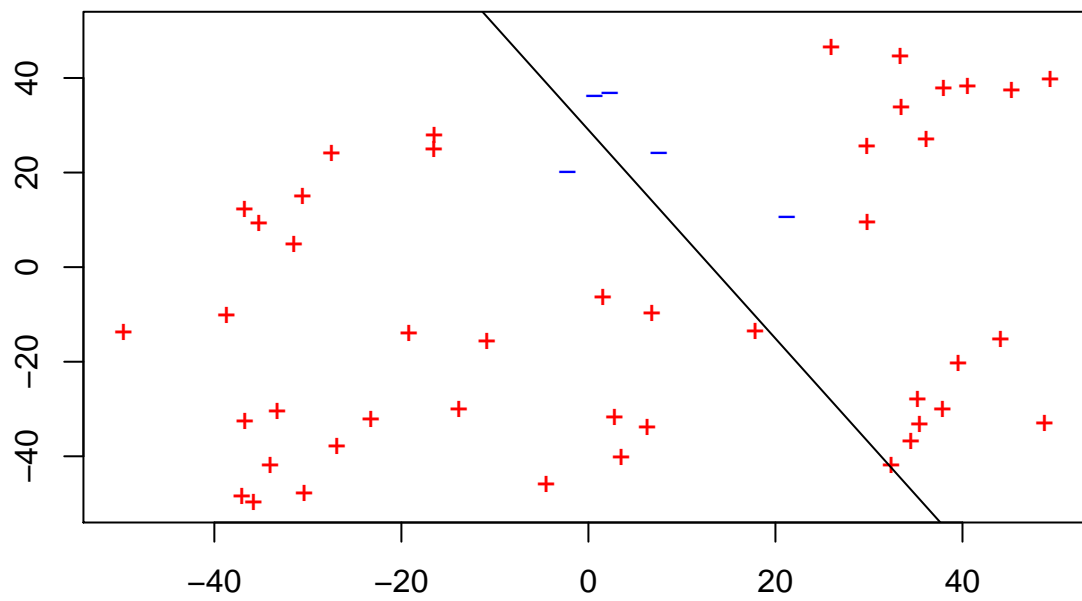
```
## [1] "Iteracion 2"
## [1] "Errores: 25"
```



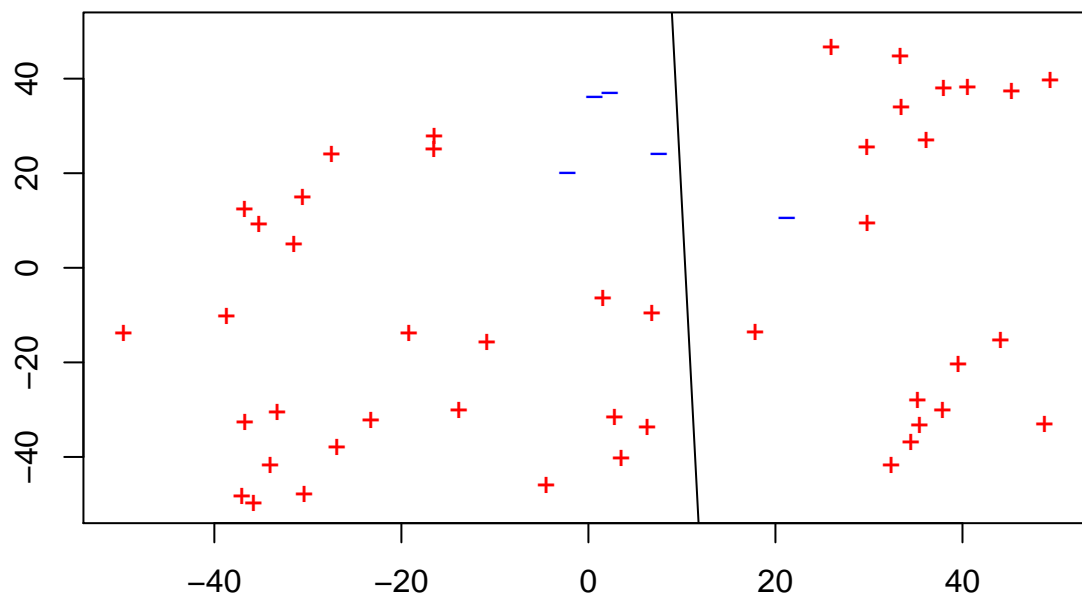
```
## [1] "Iteracion 3"
## [1] "Errores: 25"
```



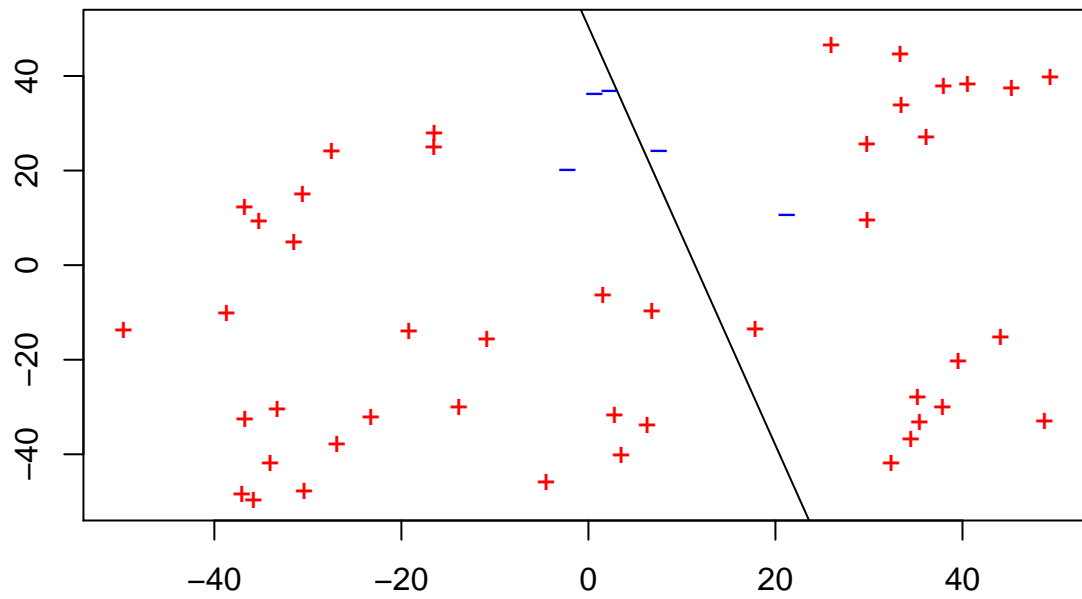
```
## [1] "Iteracion 4"
## [1] "Errores: 22"
```



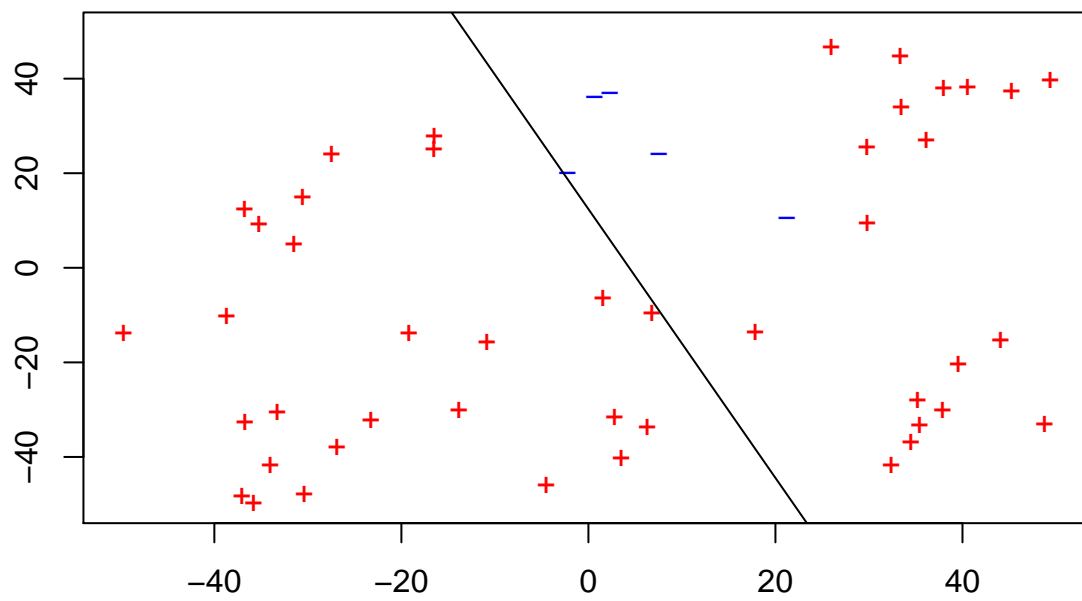
```
## [1] "Iteracion 5"
## [1] "Errores: 27"
```



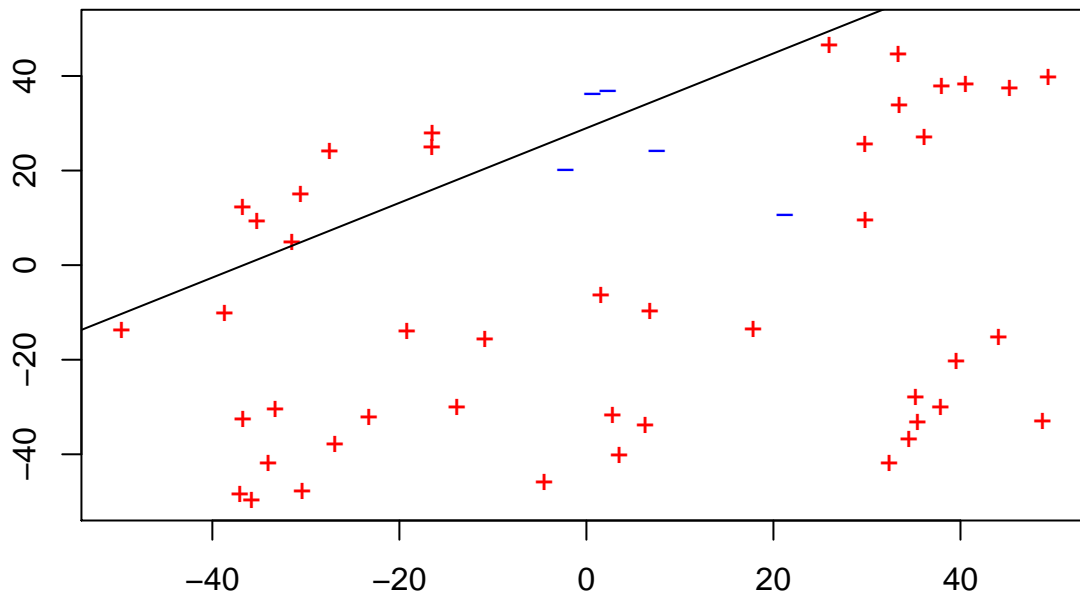
```
## [1] "Iteracion 6"
## [1] "Errores: 26"
```

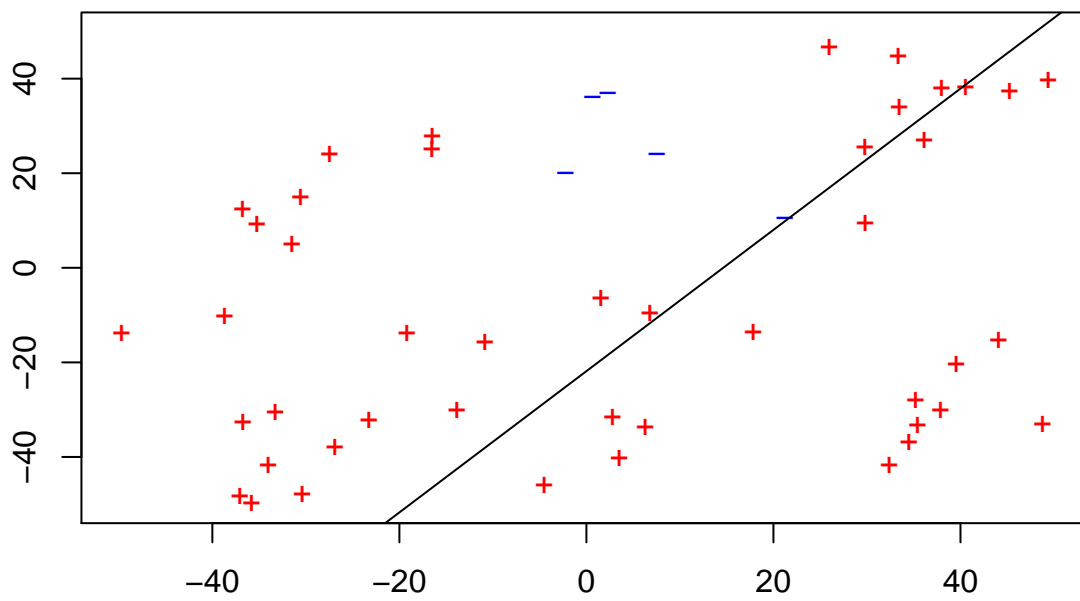
```
## [1] "Iteracion 7"
## [1] "Errores: 27"
```



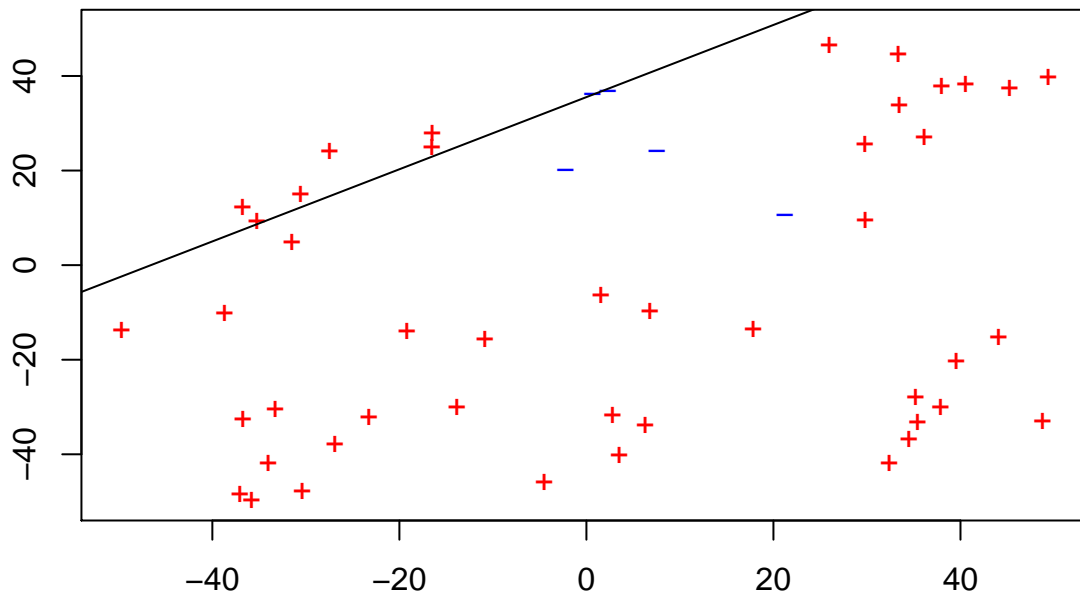
```
## [1] "Iteracion 8"
## [1] "Errores: 26"
```



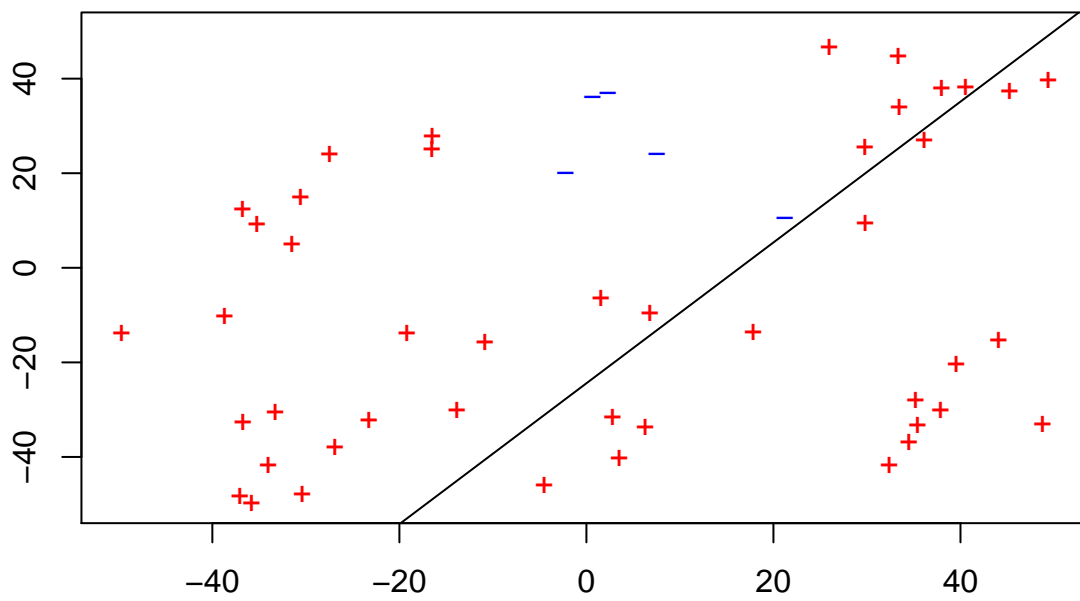
```
## [1] "Iteracion 9"
## [1] "Errores: 22"
```



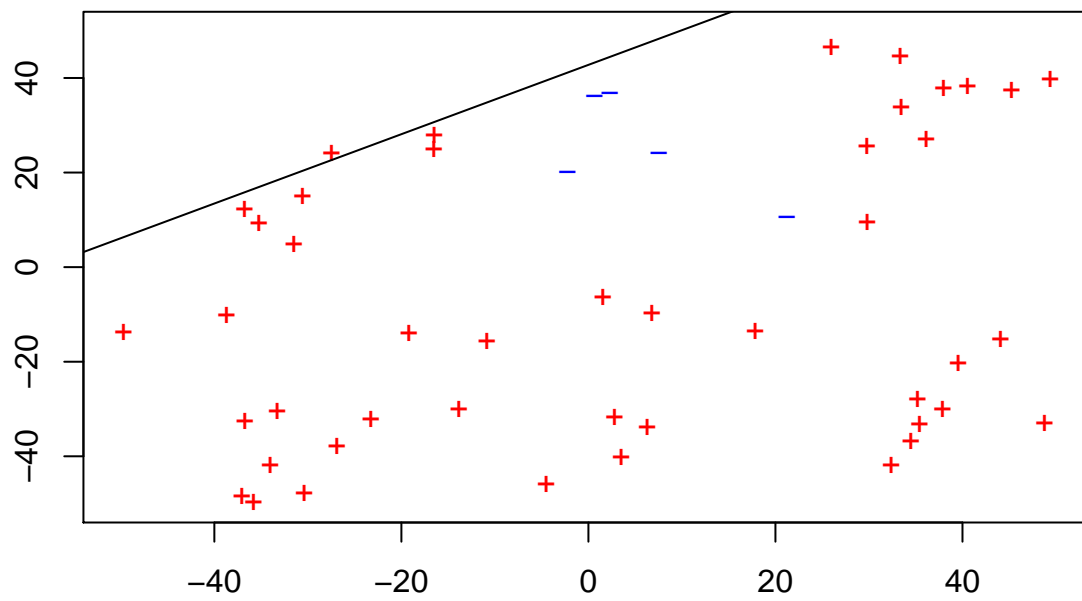
```
## [1] "Iteracion 10"
## [1] "Errores: 20"
```



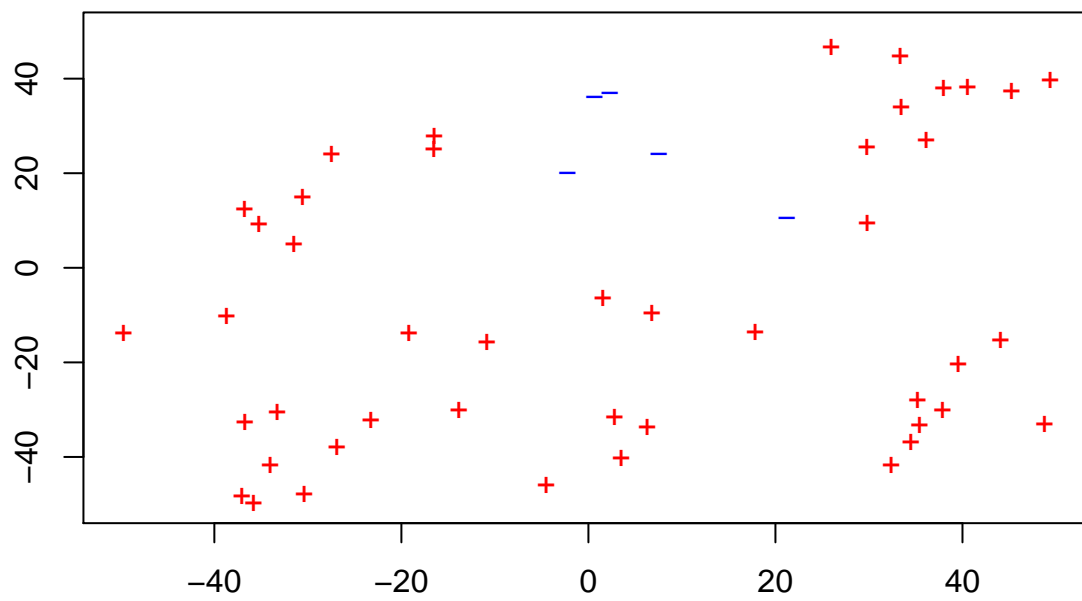
```
## [1] "Iteracion 11"
## [1] "Errores: 17"
```



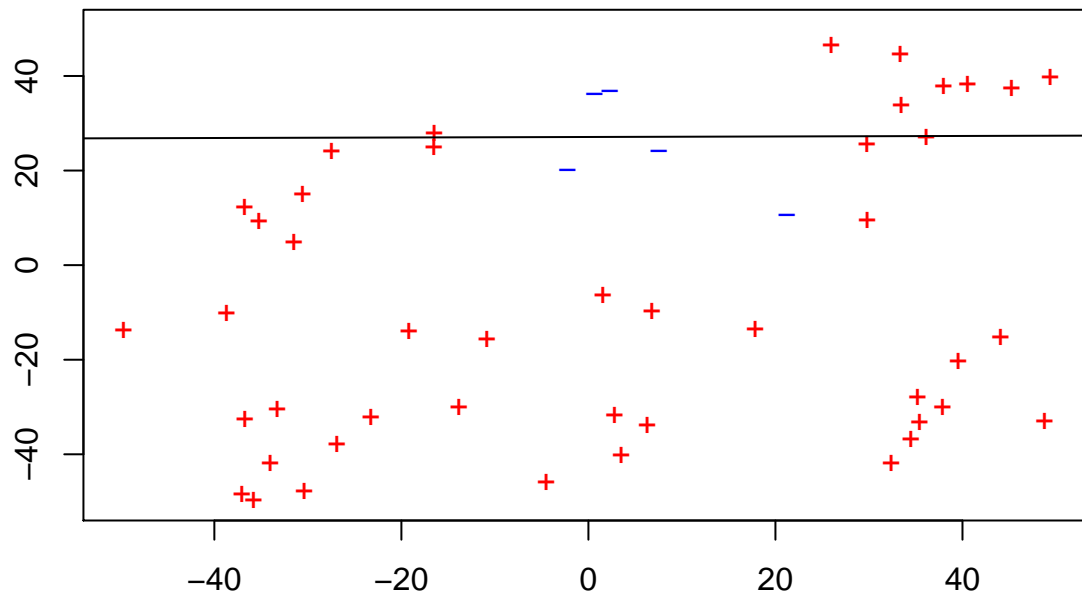
```
## [1] "Iteracion 12"
## [1] "Errores: 20"
```



```
## [1] "Iteracion 13"
## [1] "Errores: 17"
```



```
## [1] "Iteracion 14"
## [1] "Errores: 9"
```



```
## [1] "Iteracion 15"  
## [1] "Errores: 15"
```

```
print(resultado$Errores)
```

```
## [1] 15
```

A la vista de estos resultados, proponemos la siguiente mejora: guardar el mejor resultado en una variable y devolverlo.

```
ajusta_PLA_MOD <- function(datos, label, max_iter, vini){
  cambio <- TRUE
  w <- vini
  iteraciones <- 0
  errores <- 0
  X <- cbind(1,datos)
  mejor <- vini
  mejor_errores <- nrow(datos)

  while (cambio && iteraciones < max_iter){
    cambio <- FALSE
    errores <- 0
    for (i in 1:nrow(datos)){
      x_i <- as.numeric(X[i,])
      prodEscalar <- crossprod(w,x_i)
      if (sign(prodEscalar) != label[i]){
        cambio <- TRUE
        errores <- errores + 1
        w <- w + label[i] * x_i
      }
    }
    if (errores < mejor_errores){
      mejor <- w
      mejor_errores <- errores
    }
    iteraciones <- iteraciones + 1
  }

  resultado <- list("Peso inicial" = vini , "Pesos" = mejor,
                  "Iteraciones" = iteraciones, "Errores" = mejor_errores,
                  "Recta" = c(-w[1]/w[3], -w[2]/w[3]) )
  return (resultado)
}
```

Si lo ejecutamos con el mismo conjunto de antes nos devuelve

```
resultado <- ajusta_PLA_MOD(muestra.uniforme.circular[,c("X","Y")], etiquetas, 15, vini)
print(resultado$Errores)
```

```
## [1] 9
```

Regresión lineal

Ejercicios 1 y 2

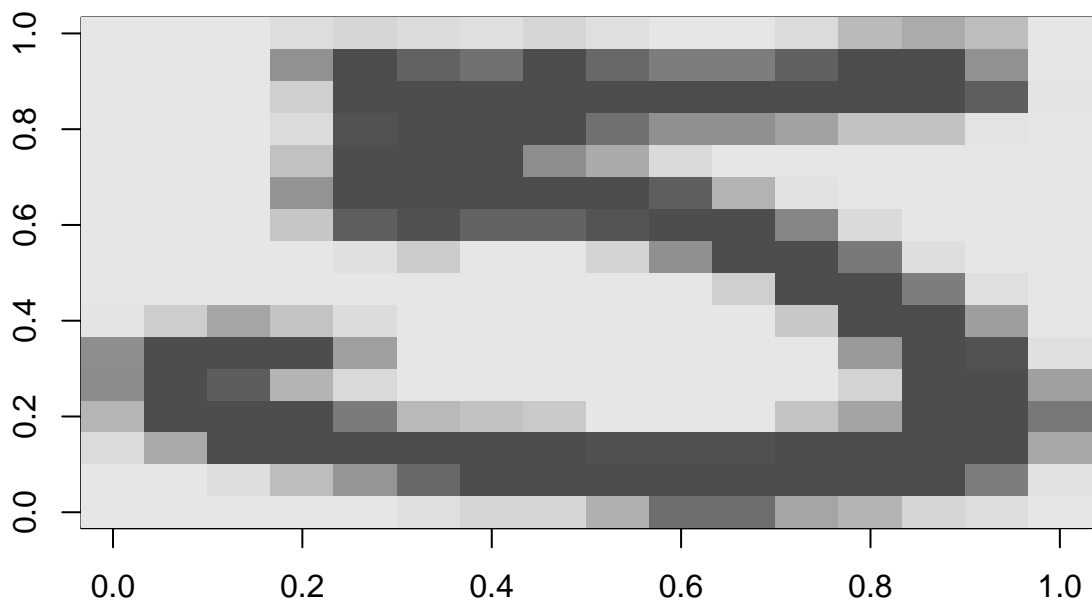
Leeremos los números desde el fichero de entrenamiento y guardaremos las instancias de 1's y 5's en sendas listas de matrices.

```
archivo <- read.csv("zip.train", header = FALSE, sep = " ")
colnames(archivo)[1] <- "ID"
archivo["V258"] <- NULL

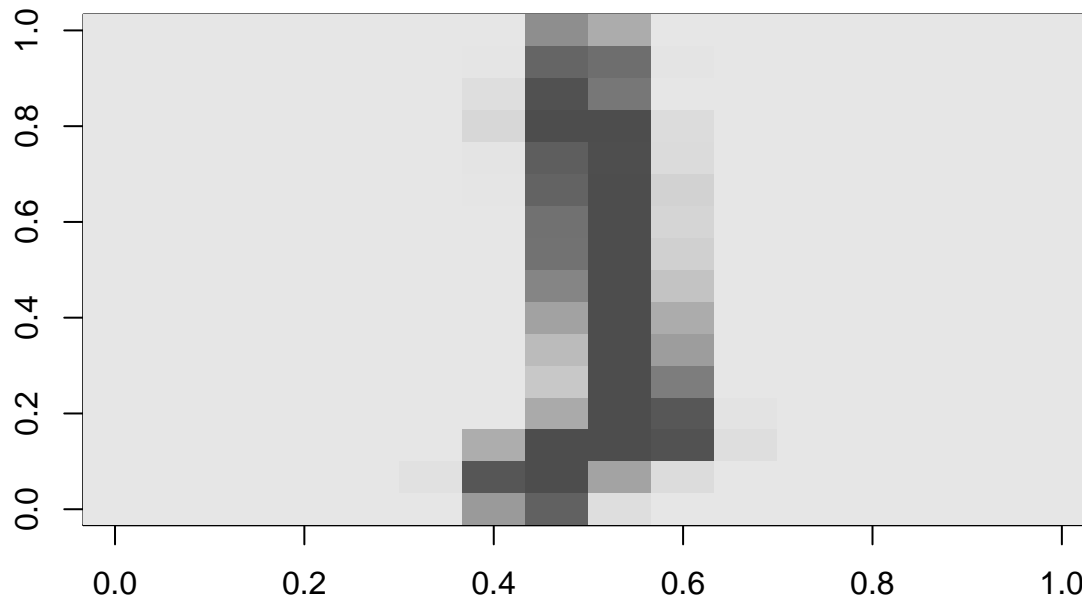
numeros_5 <- subset(archivo, ID == 5)
numeros_5["ID"] <- NULL
numeros_5 <- data.matrix(numeros_5)

numeros_1 <- subset(archivo, ID == 1)
numeros_1["ID"] <- NULL
numeros_1 <- data.matrix(numeros_1)

numeros_5_matrix <- lapply(split(numeros_5, 1:nrow(numeros_5)),
                           function(m) {
                               mtx <- matrix(0.5*(1-m), nrow = 16, ncol = 16)
                               return(mtx[,16:1])
                           })
image(numeros_5_matrix[[1]], col = gray.colors(256))
```



```
numeros_1_matrix <- lapply(split(numeros_1, 1:nrow(numeros_1)),
                           function(m) {
                               mtx <- matrix(0.5*(1-m), nrow = 16, ncol = 16)
                               return(mtx[,16:1])
                           })
image(numeros_1_matrix[[50]], col = gray.colors(256))
```



Ejercicio 3

Para cada número, calcularemos su media y su simetría vertical.

```
medias_1 <- lapply(numeros_1_matrix, mean)
medias_5 <- lapply(numeros_5_matrix, mean)

simetriaVertical <- function(vector) {
  simetria <- 2*sum(abs(vector[1:(length(vector)/2)] - vector[(length(vector)/2+1):length(vector)]))
  return(simetria)
}

simetrias_1 <- lapply(numeros_1_matrix,function(m) -sum(apply(m,1,simetriaVertical)) )
simetrias_5 <- lapply(numeros_5_matrix,function(m) -sum(apply(m,1,simetriaVertical)) )
```

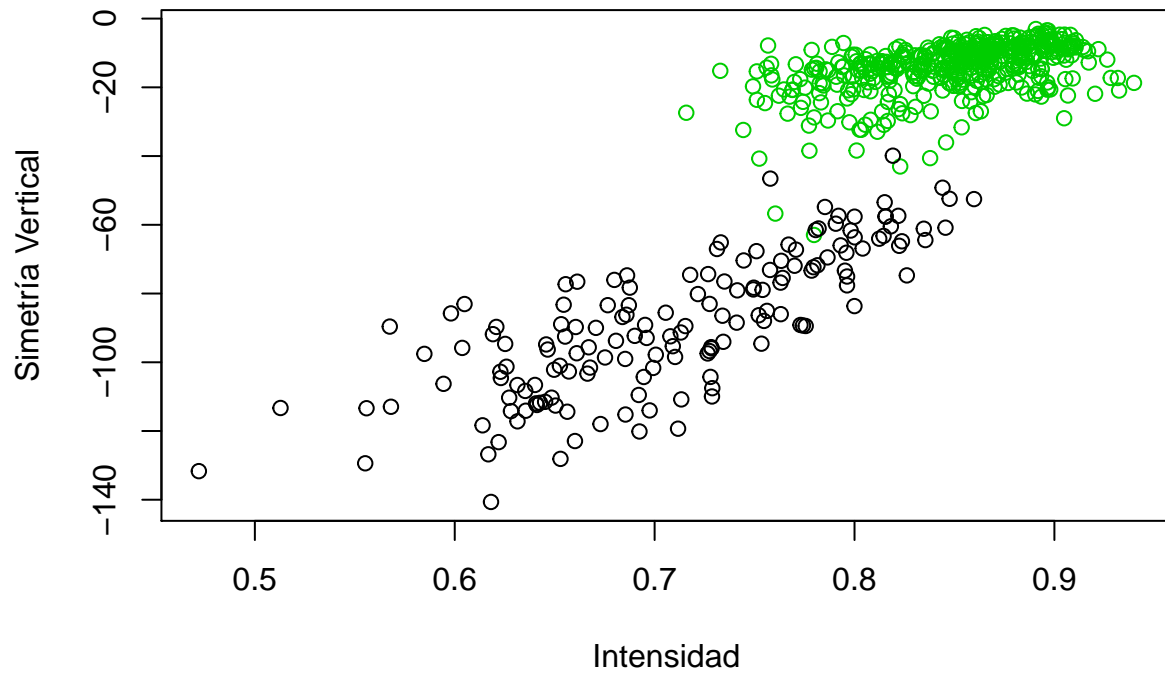
Ejercicio 4

Ahora representamos en una nube de puntos la simetría y la media de la intensidad de los números.

```
datos_1 <- data.frame(Medias = as.numeric(medias_1), Simetrias = as.numeric(simetrias_1), Etiqueta = 1)
datos_5 <- data.frame(Medias = as.numeric(medias_5), Simetrias = as.numeric(simetrias_5), Etiqueta = -1)

datos <- rbind(datos_1, datos_5)
plot(datos$Medias, datos$Simetrias, main = "Intensidad y Simetría", xlab = "Intensidad", ylab = "Simetría")
```


Intensidad y Simetría



Ejercicio 5

Aplicamos el algoritmo de regresión para obtener un clasificador lineal.

```
regres_lin <- function(datos, label, pesos = FALSE) {  
  
  if (is.null(ncol(datos)))  
    X <- cbind(1, as.numeric(datos))  
  else  
    X <- cbind(1, data.matrix(datos))  
  
  X.svd <- svd(X)  
  
  V <- X.svd$v  
  D <- X.svd$d  
  Di <- diag(ifelse(D>0.0001, 1/D, D))  
  
  Xt.X.inv <- V %*% Di^2 %*% t(V)  
  
  pseudoinversa <- Xt.X.inv %*% t(X)  
  
  w <- pseudoinversa %*% as.numeric(label)  
  
  if (is.null(ncol(datos)) || pesos == TRUE)  
    return(w)  
  else
```

```

    return(c(-w[1]/w[3], -w[2]/w[3]))
}

```

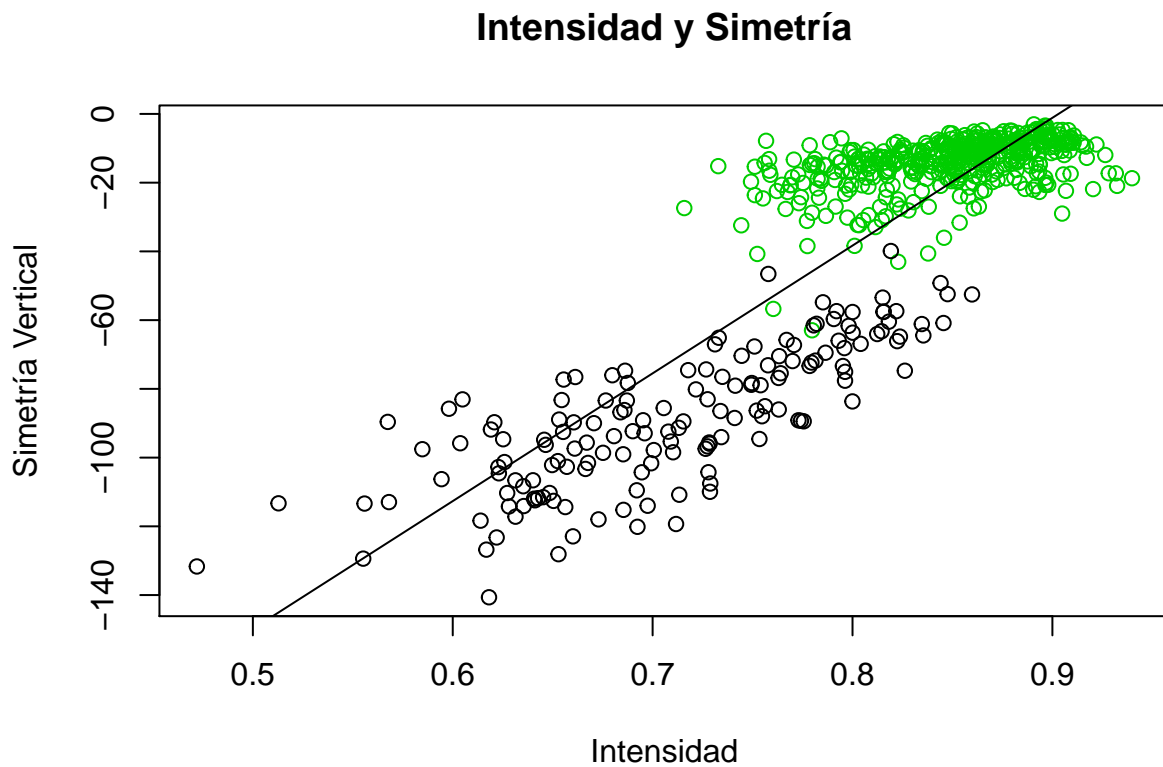
Ejercicio 6

Una vez tenemos el algoritmo, lo probamos y dibujamos la recta.

```

# muestras <- cbind(datos$Medias, datos$Simetrias)
# etiquetas <- datos$Etiqueta
muestras <- datos$Medias
etiquetas <- datos$Simetrias
recta <- regres_lin(muestras, etiquetas)
plot(datos$Medias, datos$Simetrias, main = "Intensidad y Simetría", xlab = "Intensidad", ylab = "Simetría",
      abline(recta))

```



Valoración del resultado

Ejercicio 7

Vamos a generar diversas muestras y ver cómo se comporta la regresión lineal en problemas de clasificación.

Apartado a)

Aquí analizamos el error dentro de la muestra de nuestro clasificador.

```
intervalo <- -10:10
muestras <- simula_unif(100, 2, intervalo)

errores <- 0
for (i in 1:1000) {
  f <- simulaRecta(intervalo)
  etiquetas <- sign(muestras$Y - f[2]*muestras$X - f[1])
  g <- regres_lin(muestras,etiquetas)
  etiquetas.regresion <- sign(muestras$Y - g[2]*muestras$X - g[1])

  errores <- errores + sum(etiquetas != etiquetas.regresion)/length(etiquetas)
}
errores <- errores / 1000
```

El E_{in} es 0.06589, lo cual nos da una estimación de lo bueno que puede ser nuestro clasificador dentro de la muestra.

Apartado b)

Veamos cómo se comporta el regresor lineal fuera de la muestra.

```
intervalo <- -10:10
muestras <- simula_unif(100, 2, intervalo)

errores <- 0
for (i in 1:1000) {
  f <- simulaRecta(intervalo)
  etiquetas <- sign(muestras$Y - f[2]*muestras$X - f[1])
  g <- regres_lin(muestras,etiquetas)

  muestras.nuevas <- simula_unif(1000, 2, intervalo)
  etiquetas.f <- sign(muestras.nuevas$Y - f[2]*muestras.nuevas$X - f[1])
  etiquetas.g <- sign(muestras.nuevas$Y - g[2]*muestras.nuevas$X - g[1])

  errores <- errores + sum(etiquetas.f != etiquetas.g)/length(etiquetas.f)
}
errores <- errores / 1000
```

El E_{out} es 0.061145, lo cual nos da a entender que fuera de la muestra, también se comporta bien nuestro clasificador.

Apartado c)

Ahora veamos cómo se comporta el *PLA* cuando toma como vector inicial una recta de regresión.

```
intervalo <- -10:10
muestras <- simula_unif(10, 2, intervalo)

iteraciones <- 0
iteraciones.cero <- 0
for (i in 1:100) {
  f <- simulaRecta(intervalo)
  etiquetas <- sign(muestras$Y - f[2]*muestras$X - f[1])
  w <- regres_lin(muestras,etiquetas, TRUE)
  w.cero <- rep(0,3)

  resultado <- ajusta_PLA(muestras, etiquetas, 10000, w)
  resultado.cero <- ajusta_PLA(muestras, etiquetas, 10000, w.cero)

  iteraciones <- iteraciones + resultado$Iteraciones
  iteraciones.cero <- iteraciones.cero + resultado.cero$Iteraciones
}
iteraciones <- iteraciones / 100
iteraciones.cero <- iteraciones.cero / 100
```

El número de iteraciones medio es 8.48, y el de iteraciones con vector inicial 0 es 11.88 con lo que vemos que con regresión podemos dar un vector inicial para el *PLA* que hace que éste converga de manera rápida.